EPJ Web of Conferences **245**, 04042 (2020) https://doi.org/10.1051/epjconf/202024504042
CHEP 2019

Moving the California distributed CMS XCache from bare metal into containers using Kubernetes

Edgar Fajardo[1,*], *Matevz* Tadel[1,**], *Justas* Balcas[2,***], *Alja* Tadel[1,****], *Frank* Würthwein[1,†], *Diego* Davila[1,‡], *Jonathan* Guiang[1,§], and *Igor* Sfiligoi[1,¶]

[1]University of California San Diego, 9500 Gilman Dr, La Jolla, CA 92093
[2]California Institute of Technology, 1200 E California Blvd, Pasadena, CA 91125

Abstract. The University of California system maintains excellent networking between its campuses and a number of other Universities in California, including Caltech, most of them being connected at 100 Gbps. UCSD and Caltech Tier2 centers have joined their disk systems into a single logical caching system, with worker nodes from both sites accessing data from disks at either site. This successful setup has been in place for the last two years. However, coherently managing nodes at multiple physical locations is not trivial and requires an update on the operations model used. The Pacific Research Platform (PRP) provides Kubernetes resource pool spanning resources in the science demilitarized zones (DMZs) in several campuses in California and worldwide. We show how we migrated the XCache services from bare-metal deployments into containers using the PRP cluster. This paper presents the reasoning behind our hardware decisions and the experience in migrating to and operating in a mixed environment.

1 Introduction

New ventures in data access and data placing have taken place in preparation for the High Luminosity Large Hadron Collider (HL-LHC). In the recent past the Anydata, Anytime, Anywhere (AAA) [1, 2] project showed that the CMS Computing model could be changed to one where the data could be separated from the location of the computing resources. Since data could be accessed from the Wide Area Network (WAN), where and how to place it for remote and local access became an important question. If the same data is repeatedly accessed from proximal resources efficiencies can be gained from caching the subsequent remote reads. XCache [3], an XRootD [4] based caching technology was created to exploit these efficiencies. XRootD servers allow for a tree based structure (sometimes referred to as a federation) in which the bottom level leaves are servers holding data and the upper level leaves are called

*e-mail: emfajard@ucsd.edu
**e-mail: mtadel@ucsd.edu
***e-mail: jbalcas@caltech.edu
****e-mail: amraktadel@ucsd.edu
†e-mail: fkw@ucsd.edu
‡e-mail: didavila@ucsd.edu
§e-mail: jguiang@ucsd.edu
¶e-mail: isfiligoi@sdsc.edu

redirectors. Once a client requests a file to a redirector it asks its children if they have the file. If one of them has it, the client is redirected to that server. If not, the client is redirected to the next branch above, such that leaves on a different parallel branch of the tree can be queried for the file. In this model clients are exposed to latency behind file access to remote servers (for example a job running at Caltech could end up redirected to a server in Switzerland). XCache hides this problem as it sits between a client and a federation. The cache fetches files from the federation, storing them locally. For the CMS LHC-Run-II, UCSD scale tested and deployed a multi node federated XCache (i.e federated implies the cache is presented as one logical entrypoint to the client, but it is made out of several servers) that could serve a limited part of the analysis namespace [5]. We refer to this deployment as the SoCal cache.

In this project we show how two grid sites (Caltech and UCSD) deployed a multi-site logical XCache, how data analytic drove the hardware specifications and specification of namespace, and finally how Kubernetes was chosen to allow for fast deployments of this newly built infrastructure.

2 Namespace and Working Set

The driving cost for deploying a cache is the disk. Hence, the most efficient use of resources in a cache is to choose the namespace in a way that maximizes the amount of jobs that can use it while minimizing the disk purchased. Using the CERN-based analytics database of past CMS data access, we concluded that the namespace of choice would be MINIAOD (see figure 1).

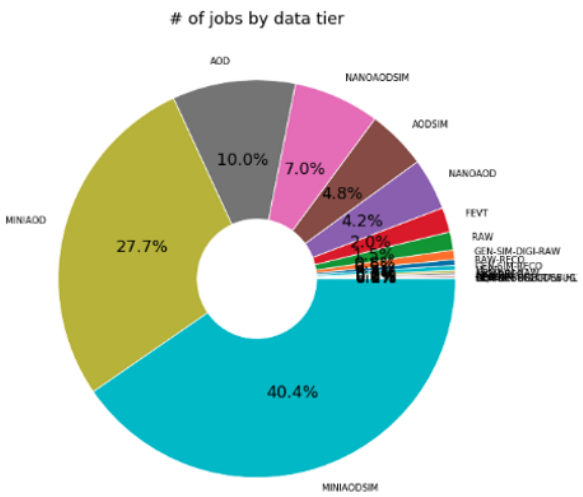

Figure 1. Distribution of Analysis Jobs in CMS in 2019. Data obtained from MONIT [6].

In order to physically store all the MINIAOD and MINIAODSIM data from CMS in the cache, 7.5 Petabytes of disk space would be needed (Table 1). However, a more efficient approach is to provision space for only the subset of data that is actually accessed. The working set is defined as the size of the unique set of files from the MINIAOD namespace accessed by jobs in a given time period. The total size of the files in the working set serves as a guide for the disk space needed to store all accessed files in a given month. An upper and lower bound for this metric can be obtained as follows.

- **Global working set**: takes into account the access at all sites. It is calculated using the data of the global pool monitoring system [7].

Table 1. Size of all datasets in different MINIAOD Data tiers. Data obtained using the CMS Data Aggreration Service DAS [8].

Data Tier	Size (PB)
/*/*/MINIAOD	2.92
/*/*/MINIAODSIM	4.6

- **SoCal working set**: only considers access at the UCSD and Caltech sites. It is calculated using the XRootD monitoring data of the SoCal cache [5].

The two monitoring sources have different data access granularities (dataset vs file). In the case of the global pool monitoring, a record stores the dataset information of a file accessed. In XRootD monitoring, the actual file information is stored. This granularity plays an important role when the working set is calculated. When using the dataset granularity a job accessing a file will trigger the addition of the whole dataset to the working set. In file granularity it would only add the accessed file to the working set. It is common that an analysis task will access an entire dataset (as opposed to a few files of the dataset). Some workflows that use Monte Carlo (MINIAODSIM) data will only read parts of the dataset until they have enough statistics.

In figure 2, the global working set goes from 1.2 PB to 2 PB over the course of a year. Each data point shows the size of the working set of the four previous weeks. In figure 3 the SoCal working set can be seen. A bar in this histogram shows the number of months in which the working set size, at the x-axis, has been observed. In this case the access to detector data and Monte Carlo-generated data are separated so they must be summed to show the total working set size. For example, in October 2019 the total working set size was 451TB.

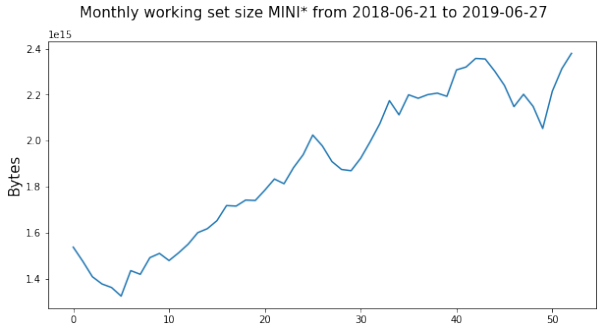

Figure 2. Monthly global working set size for analysis jobs using MINIAOD and MINIAODSIM data tiers. Data obtained from MONIT [6].

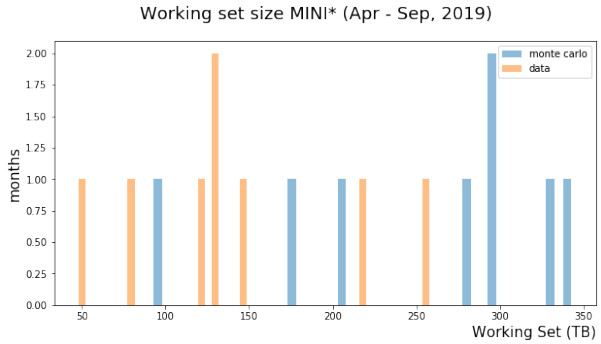

Figure 3. Monthly SoCal working set size for analysis jobs using MINIAOD and MINIAODSIM data tiers. Data obtained from MONIT [6].

Table 2. Hardware deployment specification for SoCal cache

	UCSD	**Caltech**
Nodes	11 (+1 JBOD[1])	2
Disk Capacity per node	12 x 2TB = 24TB (+ 48 x 11TB)	30 x 6TB = 180TB
Network Card per node	10 Gbps (+ 40 Gbps)	40 Gbps
Total Disk Capacity	264 TB (+ 528 TB) = 792 TB	360 TB*
Total	792TB + 360TB = 1,152TB	

3 Hardware Provisioning and Deployment

Based on the calculations of section 2, the amount of disk space needed for the SoCal cache was determined. The current hardware deployment is summarized in Table 2. Figure 2 shows the working set steadily growing therefore the infrastructure must be able to scale up horizontally.

3.1 Kubernetes

The SoCal cache is installed in heterogenous servers, mostly reused from other projects. The cache is operated according to a Development Operations model (DevOps) to cope with the heterogeneity, multi-institution deployment and rapid development of new features in XRootD. In this model there is quick turnaround between development and deployment of new software versions without the intervention of the site system administrators.

The cache services are deployed using Kubernetes. The building block of Kubernetes are pods, which are instantiated services of containers. The Open Science Grid (OSG) [9] builds new XRootD RPMs as new versions become available and maintains containers out of each new version. Moreover, OSG also provides specialized containers for the CMS XCache deployments on top of the XRootD generic ones.

DevOps model allows for deployment of new containers in minutes and to roll them back to older versions just as quickly. For the caches to work, Kubernetes functionalities like secrets and volume claims were exploited. The first one (secrets) manages the certificates of each cache and the latter one allows pods to securely (without other pods interfering) access the disk partitions of the host they are running on. This allows for other pods to be deployed in the same servers while maintaining isolation and quality of service.

There are potential frictions with this deployment model. At its core Kubernetes deployment model is based on stateless micro services, on the other hand the cache is stateful (the state is the cached files on local disk). Volume claims (local) allow to bridge this gap. They keep the state of the cache between pods restarts allowing for smooth changes of container versions. To obtain the best performance, pods did not use the Kubernetes virtualized network, instead the cache pods were allowed to bind directly to the hosts network. A future line of work is understanding what are the consequence of using this network capability. There is no current evidence of performance difference between cache services running in Kubernetes versus ones deployed in a standard (bare-metal) model. This is an area the authors would like to explore further in future work.

[1]JBOD stands for Just a bunch of disks. XRootD stores data on each of them separately without any raid setup. This is acceptable for caching since if one disk is offline the cache is still available and the data loss can be retrieved after from the federation

3.2 Monitoring

Currently, there are three sources of monitoring data surrounding this service.

1. The job monitoring data from the global pool that can be accessed through MONIT [6] (i.e. which dataset was accessed by each job).

2. The XRootD monitoring data from the XCache servers accessible through MONIT at CERN and GRACC [10].

3. The hardware probes installed in the servers.

The first source gives a view of the usage of the SoCal cache deployment from the jobs perspective and computes metrics like failure rate, average read speed, and CPU efficiency. The second source contains information about the data access from the cache perspective and calculates metrics like total data delivered, working set size, unique reads, etc. The third source is the least granular: hardware server metrics. The bare metal hosting the cache report metrics such as: system load, inbound and outbound network traffic and percentage of disk used.

Making aggregations of this monitoring data over large periods of time is of great interest for the operators of this service and could be time consuming and sometimes impossible given the data retention policy of the current monitoring systems. Based on these observations, Monicron was born as part of the TUDA project [11]. Monicron is a new monitoring system designed to calculate and store aggregations of complex metrics from both the XRootD and the Global pool jobs monitoring data over defined intervals of time. The ultimate goal of Monicron is to provide a simplified view of the health and performance of the cache. The development of this system is complete and it is anticipated to be fully operational at the time of publication of the present paper.

4 Cache Performance

An experiment was setup to evaluate the benefits of using a cache infrastructure versus other infrastructure. For one month Caltech's jobs accessed the MINIAOD data using local Hadoop (if files were available) or remote reads (AAA federation) while UCSD jobs kept accessing this data using the cache. The result being the average read time for jobs at Caltech was five times higher than using the cache, see figure 4. This result reassures as that the deployment provides value to the scientific computing in the way of better efficiency.

5 Conclusions and Future Work

By rapid allocation and deployment (via Kubernetes) of a caching system, CMS analyses efficiency can be greatly improved. Data analytic drove the hardware decisions and saved up to six petabytes of disk space that would otherwise had to be purchased. The future direction on caching deployments will be three legged. The first direction is testing different algorithms for deciding what to write and to delete to the local cache hopefully profiting of the information richness of CMS logical file names. The second direction is to develop consistency checks at the cache level (what is stored in the cache can be corrupted, over time, disk errors or even network errors). The third direction is an in depth studying of the overall impact of the deployment model of containers versus bare metal in the performance of the caches and the efficiency gains of the DevOps model.

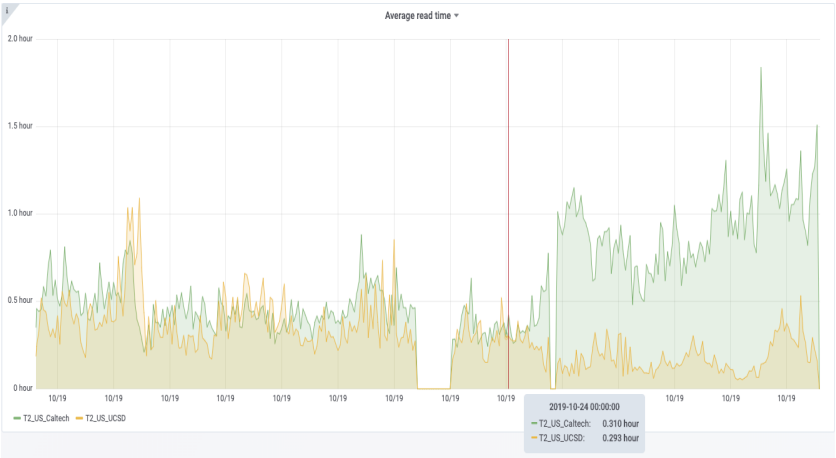

Figure 4. Average total read time for analysis jobs using MINIAOD at Caltech and UCSD. The X-axis are the dates and in the Y-axis is the average read time for in hours for a job. The red-line is when Caltech experimented not reading from cache but reading from the AAA federation.

Acknowledgement

The authors would like to thank the different funding agencies for this work, in particular the National Science Foundation through the following grants: OAC-1541349, MPS-1148698, OAC-1836650, MPS-1624356, OAC-1826967.

References

[1] K.B. for the CMS Collaboration, Journal of Physics: Conference Series **513**, 042005 (2014)

[2] L. Bauerdick, D. Benjamin, K. Bloom, B. Bockelman, D. Bradley, S. Dasu, M. Ernst, R. Gardner, A. Hanushevsky, H. Ito et al., Journal of Physics: Conference Series **396**, 042009 (2012)

[3] L. Bauerdick, K. Bloom, B. Bockelman, D. Bradley, S. Dasu, J. Dost, I. Sfiligoi, A. Tadel, M. Tadel, F. Wuerthwein et al., Journal of Physics: Conference Series **513** (2014)

[4] A. Dorigo, P. Elmer, F. Furano, A. Hanushevsky, WSEAS Transactions on Computers **1** (2005)

[5] E. Fajardo, A. Tadel, M. Tadel, B. Steer, T. Martin, F. Würthwein, Journal of Physics: Conference Series **1085**, 032025 (2018)

[6] A. Aimar, A. Corman, P. Andrade, J. Fernandez, B. Bear, E. Karavakis, D. Kulikowski, L. Magnoni, EPJ Web of Conferences **214**, 08031 (2019)

[7] J. Balcas, S. Belforte, B. Bockelman, D. Colling, O. Gutsche, D. Hufnagel, F. Khan, K. Larson, J. Letts, M. Mascheroni et al., Journal of Physics: Conference Series **664**, 062031 (2015)

[8] M. Giffels, Y. Guo, V. Kuznetsov, N. Magini, T. Wildish, Journal of Physics: Conference Series **513**, 042052 (2014)

[9] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein et al., Journal of Physics: Conference Series **78**, 012057 (2007)

[10] K. Retzke, D. Weitzel, S. Bhat, T. Levshina, B. Bockelman, B. Jayatilaka, C. Sehgal, R. Quick, F. Wuerthwein, Journal of Physics: Conference Series **898**, 092044 (2017)

[11] J. Guiang, *jkguiang/tuda: Zenodo release* (2020), `https://doi.org/10.5281/zenodo.3636224`